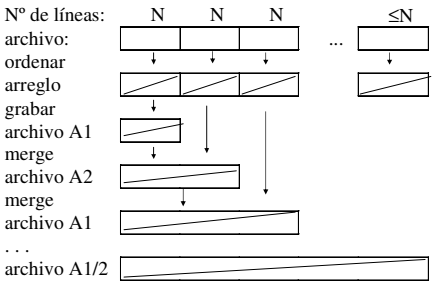
Problema. Ordenar un archivo (suponiendo que sólo N líneas caben en memoria)
Solución. Algoritmo de "Cascada".



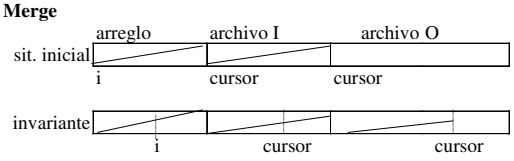

```
import java.io.*;
class Sortfile
{
//arreglo para N líneas
protected static int N=100, n=0;
protected String[]linea=new String[N];

public void main(String[]args)throws IOException
{
//grabar archivo auxiliar vacio
PrintWriterW B=new PrintWriter(
  new FileWriter("A2.txt"));
B.close();
//abrir archivo y nombrar archivos auxilares
BufferedReader A=new BufferedReader(
  new FileReader(args[0]));
String in="A2.txt", out="A1.txt";
```

```
//repetir hasta terminar archivo
while(true)
{
  //leer n líneas de archivo(max N) en arreglo
  n=leerLineas(A,linea,N);
  if(n==0) break;

  //ordenar arreglo de n líneas
  quicksort(linea,0,n-1);

  //merge de arreglo y archivo in(resultado en out)
  merge(linea,n,in,out);

  //intercambiar rol de archivos
  String aux=in; in=out; out=aux;
}
A.close();
U.println("resultado: archivo "+in);
}
```

```
//Leer desde (posición del cursor de) archivo
//un máximo de N líneas.
//Entregar también nº de líneas leídas.

static public int leerLineas
(BufferedReader archivo, String[]lineas, int N)
throws IOException
{
  int n=0;
  String linea;
  while((linea=archivo.readLine())!=null){
    if(n>=N) U.abortar("max de lineas ="+N);
    lineas[n++] = linea;
  }
  return n;
}
```

```
//merge de arreglo y archivo ordenados
static public void merge
(String[]x,int n,String input,String output)
throws IOException
{
 PrintWriter O=new PrintWriter(
   new FileWriter(output));
 BufferedReader I=new BufferedReader(
    new FileReader(input));
```

Merge




```
 //obtener primeros elementos
 int i=0; //indice primer elemento
 String L=I.readLine();//primera línea

 //repetir hasta terminar arreglo y archivo
 while(i<n || L!=null)//!(i>=n && L==null)
 {
  //grabar menor (o el que quede) y avanzar
  if(i<n && (L==null || x[i].compareTo(L)<=0)){
    O.println(x[i]);//grabar elemento
    ++i;            //siguiente índice
  }else{
    O.println(L);  //grabar línea
    L=I.readLine();//sgte línea
  }
 }
 I.close(); O.close();
}}
```

Archivos de Acceso Directo ("arreglos" en disco)

índice	posición	registro
0	0	
1	L	
2	2L	
. . .	. . .	. . .
N-1	(N-1)L	

Largo Fijo = L

Archivos de texto	**Archivos Acceso Directo**
acceso secuencial	acceso directo (por posición)
líneas de largo variable (delimitadas por newline)	registros de largo fijo (sin delimitadores)
lectura o escritura	lectura y escritura
grabación en caracteres	grabación en binario
editable	no editable

clase RandomAccessFile ("RAF")

Método	**Significado**	**ejemplo**
RandomAccessFile (String x,String y)	Abre archivo de nombre x (y="rw"(read/write) o "r"). Cursor al comienzo.	RAF A =new RAF ("a.bin","rw");
void seek(long x)	posiciona cursor en byte ubicado en posición x>=0	A.seek(x);
void writeTipo(Tipo x)	escribe x, y avanza cursor en tamaño del Tipo (Int, Double, Char)	A.writeInt(10); escribe 10 y avanza 4 bytes
Tipo readTipo()	lee y entrega valor del tipo indicado (y avanza cursor)	A.readDouble() lee double y avanza 8 bytes
long getFilePointer()	entrega posición del cursor	A.getFilePointer()
long length()	entrega tamaño en bytes del archivo	A.length()

ordenar archivo de acceso directo

```
import java.io.*;
class SortRAF{
protected static int LargoReg;//en bytes
public static void main(String[]args)
throws IOException{
  //chequear nº de parámetros
  if(args.length!=2)
    U.abortar("uso:java SortRAF archivo lreg");
  //obtener archivo y largo en bytes de registros
  RAF archivo=new RAF(args[0],"rw" );
  LargoReg=Integer.parseInt(args[1]);
  //calcular nº de registros
  final int n=(int)archivo.length()/LargoReg;
  //ordenar con algoritmo para arreglos
  quicksort(archivo,0,n-1);
}
```

```
public static void quicksort
(RandomAccessFile archivo,int ip,int iu)
throws IOException
{
  //caso base: 0 o 1 registro
  if(ip>=iu) return;

  //particionar archivo
  int i=particionar(archivo,ip,iu);

  //ordenar primera parte
  quicksort(archivo,ip,i-1);

  //ordenar segunda parte
  quicksort(archivo,i+1,iu);
}
```

```
public static int particionar
(RAF archivo,int ip,int iu)
throws IOException{
  String pivote=leerRegistro(archivo,ip);
  int i=ip;
  for(int j=ip+1; j<=iu; ++j)
    if(leerRegistro(archivo,j).compareTo(pivote)<0)
     intercambiar(archivo,++i,j);
  intercambiar(archivo,ip,i);
  return i;
}
public static void intercambiar
(RAF archivo,int i,int j)throws IOException{
  String reg_i=leerRegistro(archivo,i),
         reg_j=leerRegistro(archivo,j);
  escribirRegistro(archivo,reg_i,j);
  escribirRegistro(archivo,reg_j,i);
}
```

```
//leer registro ubicado en índice
static public String leerRegistro
(RAF archivo,int indice)throws IOException{
  archivo.seek(indice*LargoReg);//cursor
  String registro="";
  for(int i=1; i<=LargoReg/2; ++i)
    registro += archivo.readChar();
  return registro;
}
//grabar registro en índice
static public void escribirRegistro
(RAF archivo,String registro,int indice)
throws IOException{
  archivo.seek(indice*LargoReg);//cursor
  for(int i=0; i<registro.length(); ++i)
    archivo.writeChar(registro.charAt(i));
}
```

Problema. Reescribir búsqueda binaria para RAF

```
static public int indice
(Comparable x,Comparable[]y,int ip,int iu)
{
  //devolver -1 si x no está en y(o y vacío)
  if(ip>iu) return -1; //no esta

  //comparar x con elemento que está en la mitad
  int im=(ip+iu)/2;   //indice del medio
  int c=x.compareTo(y[im]);

  //si son iguales,devolver índice de mitad
  if(c==0) return im;

  //si no, buscar en mitad del arreglo
  if(c<0) iu=im-1; else ip=im+1;
  return indice(x,y,ip,iu);
}
```

Solución

```
static public int indice
(Comparable x,RAF y,int ip,int iu)
throws IOException{
  //devolver -1 si x no está en y(o y vacío)
  if(ip>iu) return -1; //no esta

  //comparar x con elemento que está en la mitad
  int im=(ip+iu)/2;   //indice del medio
  int c=x.compareTo(leerRegistro(y,im));

  //si son iguales,devolver índice de mitad
  if(c==0) return im;

  //si no, buscar en mitad del archivo
  if(c<0) iu=im-1; else ip=im+1;
  return indice(x,y,ip,iu);
}
```

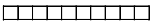
Resumen Algoritmos de Búsqueda y Ordenamiento

problema	algoritmo	O(x)	.txt	RAF	Obs
buscar	Secuencial	n	si	si	
buscar	Binaria	$\log_2 n$	no	si	ordenado
buscar	**Hashing**	1	no	si	
ordenar	Selección	n^2	no	si	
ordenar	Burbuja	n^2	no	si	
ordenar	Inserción	n^2	no	si	
ordenar	MergeSort	$n \log_2 n$	no	si	
ordenar	QuickSort	$n \log_2 n$	no	si	
ordenar	Cascada	$k N \log_2 N$	si	no	para .txt

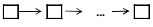
Resumen Estructuras de datos

- Formas de organización y almacenamiento de datos (en memoria)
- Sirven para representar Tipos de datos abtractos (TDAs) con múltiples valores. Ej: Stack, Queue, Diccionario, Conjunto, Polinomio, etc

arreglo



lista enlazada



árbol

Clases Stack/Queue

new Stack/Queue()
void push/enque(Object x)throws Full
Object pop/deque( )throws Empty
void reset( ) boolean empty/full( )

Tiempos de operaciones

Est de Datos	push	pop	enque	deque
arreglo	1	1	1	n
arreglo "circular"			1	1
lista enlazada	1	1	1	n
con 1º y último			1	1
"circular"			1	1
lista doble enlace			1	1

n significa O(n), es decir, depende de nº de valores
1 significa O(1), es decir, no depende de nº de valores

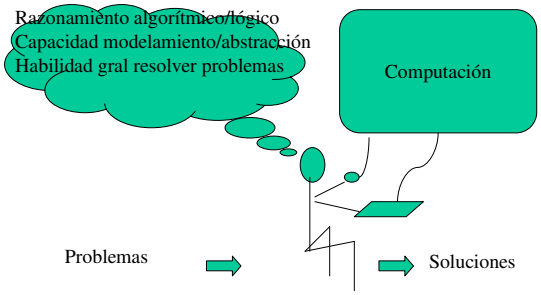
Diccionario

Object buscar(Object x)	devolver significado de palabra
boolean agregar(Object x,Object y) throws DiccLleno	agregar palabra x con significado y (false si ya existe)
boolean borrar(Object x)	borrar palabra x (false si no existe)
boolean cambiar(Object x,Object y)	cambiar significado de palabra x

Tiempos:

Estructura de datos	buscar	agregar	borrar	cambiar
arreglo ordenado	$\log_2 n$	n	n	$\log_2 n$
lista enlazada	n	n	n	n
lista doble enlace	n	n	n	n
ABB	h(altura)	h	h	h
ABB balanceado	log2n	log2n	log2n	log2n

CC1001- Computación I



Objetivos

- general
  - resolver problemas
- específico
  - escribir programas que resuelvan problemas
- de largo plazo (propósitos)
  - razonamientos algorítmico y lógico
  - capacidades de abstracción y modelamiento
  - habilidad general para resolver problemas

Metodología

- Pedagogía
  - orientación al aprendizaje
  - clases de cátedra y auxiliares centradas en problemas
  - ejercicios clases “obligatorios”
- Tecnología
  - lenguajes Java y Matlab
  - ambiente Internet/web

Contenidos

1. Fundamentos de programación
2. Programación orientada a objetos
3. Listas y tablas de valores
4. Computación numérica
5. Búsqueda y ordenamiento de información

Evaluación

- NF (nota final): 70%NC+30%NT (NC,NT ≥ 4)
- NT (nota tareas): promedio 5 tareas (1 por cap)
  Si 3≤NT<4 y NC ≥ 4, NF=I + tarea extra
- NC (nota control): prom ponderado controles

control	contenidos	semana	pond
1	1	4	20 %
2	2	8	20 %
3	3 y 4	15	20 %
examen	1 a 5	9 julio	40 %

Ejercicios

- se considera promedio de los 19 mejores
- si tiene menos de 19, NF=I + tarea(s) extra(s)
- sirven para eximirse de examen de acuerdo a la sgte tabla:

promedio c1,c2,c3	promedio ejercicios
5,5	cualquiera
5,4	≥ 2.0
5,3	≥ 2.5
5,2	≥ 3.0
5,1	≥ 3.3
5,0	≥ 3.5